2021.07.17. 교육

나정휘

https://justiceHui.github.io/

목차

- 수학적 귀납법
- 분할 정복
- 분할 정복의 예시 (1)
- 마스터 정리
- 분할 정복의 예시 (2)
- 세그먼트 트리

수학적 귀납법

수학적 귀납법

- 자연수에 대한 명제 P(n)이 모든 자연수에 대해 성립함을 증명
  - 어떤 정수 n0에 대해, n ≥ n0을 만족하는 모든 n에 대해 P(n)이 성립

- (basis step) P(n0)이 성립함을 증명
- (inductive step) P(k)가 성립하면 P(k+1)이 성립함을 증명
  - k는 n0보다 크거나 같은 정수

- P(n0) ⇒ P(n0 + 1) ⇒ P(n0 + 2) ⇒ P(n0 + 3) ⇒ …
- n ≥ n0인 모든 정수 n에 대해 P(n)이 성립

수학적 귀납법 예시

- 1 + 3 + 5 + … + (2n-1) = n^2이 성립함을 증명하자.
  - P(n) = 1 + 3 + 5 + … + (2n-1) = n^2이 성립하는가?
  - n0 = 1

- P(1)은 1 = 1^1이므로 자명
- P(k) : 1 + 3 + 5 + … + (2k-1) = k^2 성립한다고 가정하자.
  - 양변에 2k+1을 더하면
  - 1 + 3 + 5 + … + (2k-1) + (2k+1) = k^2 + 2k + 1 = (k+1)^2
  - 1 + 3 + 5 + … + (2k-1) + (2(k+1)-1) = (k+1)^2
  - 1 + 3 + 5 + … + (2(k+1)-1) = (k+1)^2 : P(k+1)

강한 수학적 귀납법

- (basis step) P(n0)이 성립
- (inductive step) $n \leq k$인 모든 n에 대해 P(n)이면 P(k+1)

강한 수학적 귀납법의 예시

- 모든 트리에는 차수가 1 이하인 정점이 있음을 증명
  - P(n) := 정점이 n개인 트리에서 성립하는가?
  - n0 = 1
- P(1) : 정점이 1개인 트리는 차수가 0인 정점이 있음
- 정점이 1..k인 트리에 모두 차수가 1 이하인 정점이 있다고 하자.
  - 정점이 k+1인 트리는 정점이 k개 이하인 1개 이상의 트리가
  - 새로 추가된 정점과 연결되어 있는 형태
  - 정점이 k개 이하인 트리에 차수가 1 이하인 정점이 있으므로
  - 정점이 k+1개인 트리에도 차수가 1 이하인 정점이 있음 : P(k+1)

질문?

분할 정복

스테이시스 필드

분할 정복

- 큰 문제를 해결하는 것은 어렵다.
  - 여러 개의 작은 문제로 쪼개서 각각 해결

```
void DivideAndConquer(InputType in, OutputType out){
    // 문제의 크기가 충분히 작은 경우 직접 해결
    if(in.size() <= Small){
        DirectSolve(in, out);
        return;
    }

    // 문제를 K개의 부분 문제로 분할함
    InputType in_small[K] = Divide(in, K);
    OutputType out_small[K];
    for(int i=0; i<K; i++){
        DivideAndConquer(in_small[i], out_small[i]);
    }
    out = Combine(out_small[0], out_small[1], ... , out_small[k-1]);
}
```

분할 정복의 시간 복잡도

- $T(N) = D(N) + sum(T(i)) + C(N)$ if $N > Small$

  $S(N)$ if $N \le Small$
  - N : 입력의 크기
  - T(N) : 입력의 크기가 N인 문제를 푸는데 걸리는 시간
  - D(N) : 크기가 N인 문제를 분할할 때 걸리는 시간
  - C(N) : 크기가 N인 문제에서 Combine에 걸리는 시간
  - S(N) : 크기가 N인 문제를 DirectSolve로 해결하는데 걸리는 시간

- 나눠지는 부분 문제의 크기를 모두 동일하게 한다면
  - $T(N) = D(N) + a*T(N/b) + C(N)$

분할 정복의 예시 1

병합 정렬

- N개의 자연수로 구성된 배열을 정렬
  - N개의 자연수를 바로 정렬하는 것은 어려움
  - N = 1이면 정렬 안 해도 됨
  - N = 2이면 쉬움
  - Small = 2로 잡고 분할 정복을 해보자!

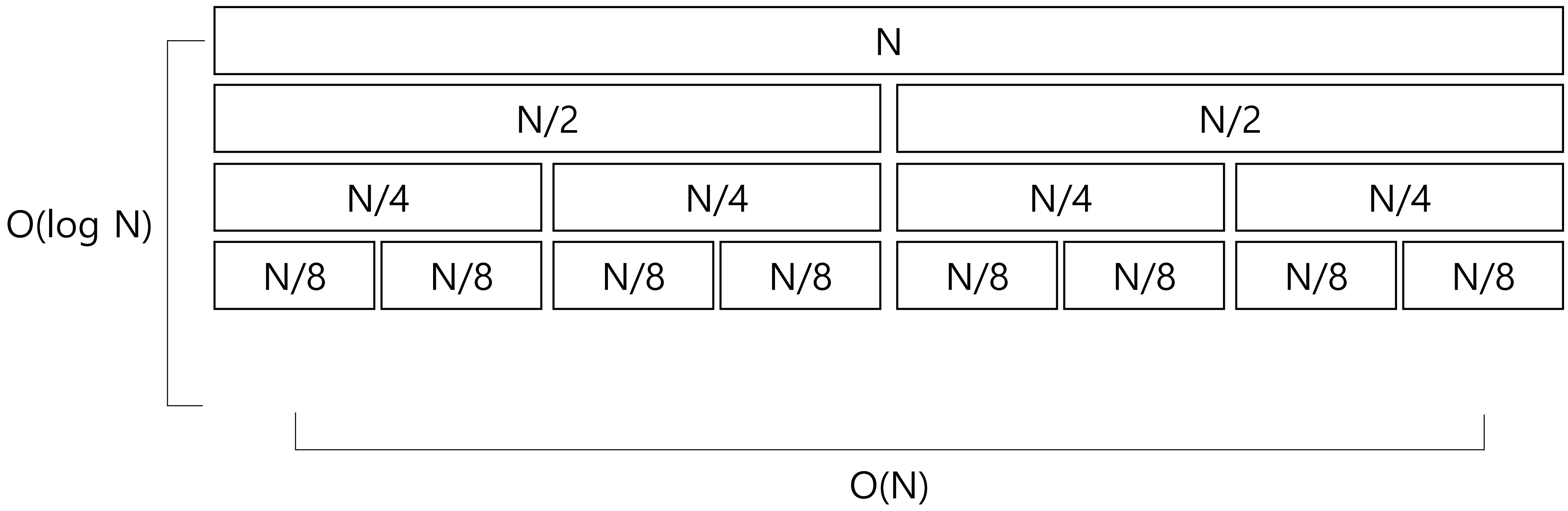
- 배열의 크기가 2보다 크면 배열을 절반으로 나눠서 각각 정렬
- 정렬된 두 배열을 합치는 건 O(N)에 가능

병합 정렬

```
const int MAX_SZ = 101010;
int Temp[MAX_SZ]; // Merge 결과 임시 저장

void Merge(int A[], int s, int m, int e){
    int i = s, j = m + 1, idx = s; // 왼쪽 배열은 s~m, 오른쪽 배열은 m+1~e
    while(i <= m && j <= e){
        if(A[i] < A[j]) Temp[idx++] = A[i++];
        else            Temp[idx++] = A[j++];
    }
    while(i <= m)       Temp[idx++] = A[i++];
    while(j <= e)       Temp[idx++] = A[j++];

    for(int k=s; k<=e; k++) A[k] = Temp[k];
}

void MergeSort(int A[], int s, int e){
    int sz = e - s + 1;
    if(sz == 1) return; // N = 1이면 정렬 안 해도 됨
    if(sz == 2){        // N = 2이면 간단한 문제
        if(A[s] > A[e]) swap(A[s], A[e]);
        return;
    }

    int m = (s + e) / 2;
    MergeSort(s, m);   // 왼쪽 정렬
    MergeSort(m+1, e); // 오른쪽 정렬
    Merge(A, s, m, e); // 정렬된 두 배열을 합치는 건 O(sz)에 가능
}

MergeSort(A, 0, N-1);
```

병합 정렬의 시간 복잡도

- T(N) = 2*T(N/2) + O(N)
  - D(N) = O(1), C(N) = O(N), S(N) = O(1)
- T(N) = O(N log N)



퀵 정렬

- N개의 자연수로 구성된 배열을 정렬
  - N = 1이라면 정렬 안 해도 됨 : Small = 1
  - 임의의 원소 x를 선택
  - x보다 작은 원소는 x 왼쪽, 큰 원소는 x 오른쪽으로 이동
  - x를 기준으로 양쪽에 대해 재귀적으로 반복

  - 배열을 2개로 나누는 건 비슷한데, 매번 나눠지는 배열의 크기가 다름

퀵 정렬의 시간 복잡도

- T(N) = T(i) + T(N-i-1) + O(N)
  - D(N) = O(N), C(N) = O(1), S(N) = O(1)
  - 왼쪽 배열의 크기가 i이면 오른쪽 배열의 크기는 N-i-1

- 최선의 경우: 동일하게 분할되는 경우
  - T(N) = 2*T(N/2) + O(N) = O(N log N)
- 최악의 경우: 0개, N-1개로 분할되는 경우
  - T(N) = T(N-1) + O(N) = O(N^2)

이분 탐색

- $T(N) = T(N/2) + O(1) = O(\log N)$
  - $T(1)$, $T(2)$, $T(4)$, $T(8)$, $T(16)$, … , $T(N)$: $O(\log N)$번

질문?

마스터 정리

마스터 정리

- T(n) = a*T(n/b) + f(n) 꼴의 점화식을 계산하는 방법 (a≥1, b>1)
  - n/b가 정수가 아닐 수도 있는데, floor나 ceil를 해도 성립함
- $T(n) = aT(n/b) + f(n)$
  - $f(n) = O\left(n^{\log_b a - \epsilon}\right)$이면 $T(n) = \Theta\left(n^{\log_b a}\right)$
  - $f(n) = \Theta\left(n^{\log_b a}\right)$이면 $T(n) = \Theta\left(n^{\log_b a} \log n\right)$
  - $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ 이면 $T(n) = \Theta\left(f(n)\right)$

마스터 정리의 확장

- f(n) = n log n이면 다항식이 아니라서 적용 불가
  - f(n) = n^k log^p(n)일 때에도 적용 가능하게 확장할 수 있다.

- $T(n) = aT(n/b) + n^k(\log n)^p$
  - $f(n) = O(n^{\log_b a - \epsilon})$이면 $T(n) = \Theta(n^{\log_b a})$
  - $a = b^k$이면 p > -1, p = -1, p < -1일 때 각각
    - $\Theta(n^{\log_b a}(\log n)^{p+1}), \Theta(n^{\log_b a} \log\log n), \Theta(n^{\log_b a})$
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$이면 p ≥ 0, p < 0일 때 각각
    - $\Theta(n^k(\log n)^p), \Theta(n^k)$

병합 정렬의 시간 복잡도

- $T(n) = a*T(n/2) + \Theta(n)$
  - $a = 2$, $b = 2$
  - $f(n) = \Theta(n^1)$
- $T(n) = \Theta(n^1 \log n) = \Theta(n \log n)$

자주 나오는 시간 복잡도

- T(n) = T(n/2) + Θ(1)   = Θ(log n)
- T(n) = T(n/2) + Θ(log n)   = Θ(log^2 n)
- T(n) = 2T(n/2) + Θ(log n)   = Θ(n)
- T(n) = 2T(n/2) + Θ(n)   = Θ(n log n)
- T(n) = 2T(n/2) + Θ(n log n) = Θ(n log^2 n)
- T(n) = 2T(n/2) + Θ(n^2)   = Θ(n^2)

질문?

분할 정복의 예시 2

배열에서의 분할 정복

- 배열 A의 구간 [s,e]에 대해 정의된 함수 f(s, e)
- f(s, e)의 최댓값/최솟값/합 등등을 구하는 문제

- 배열의 중간 지점 m = (s + e) / 2을 잡아서
- A[m]을 지나는 모든 구간을 처리하고
- [s,m-1], [m+1,e]로 나눠서 해결

- T(n) = 2*T(n/2) + f(n)

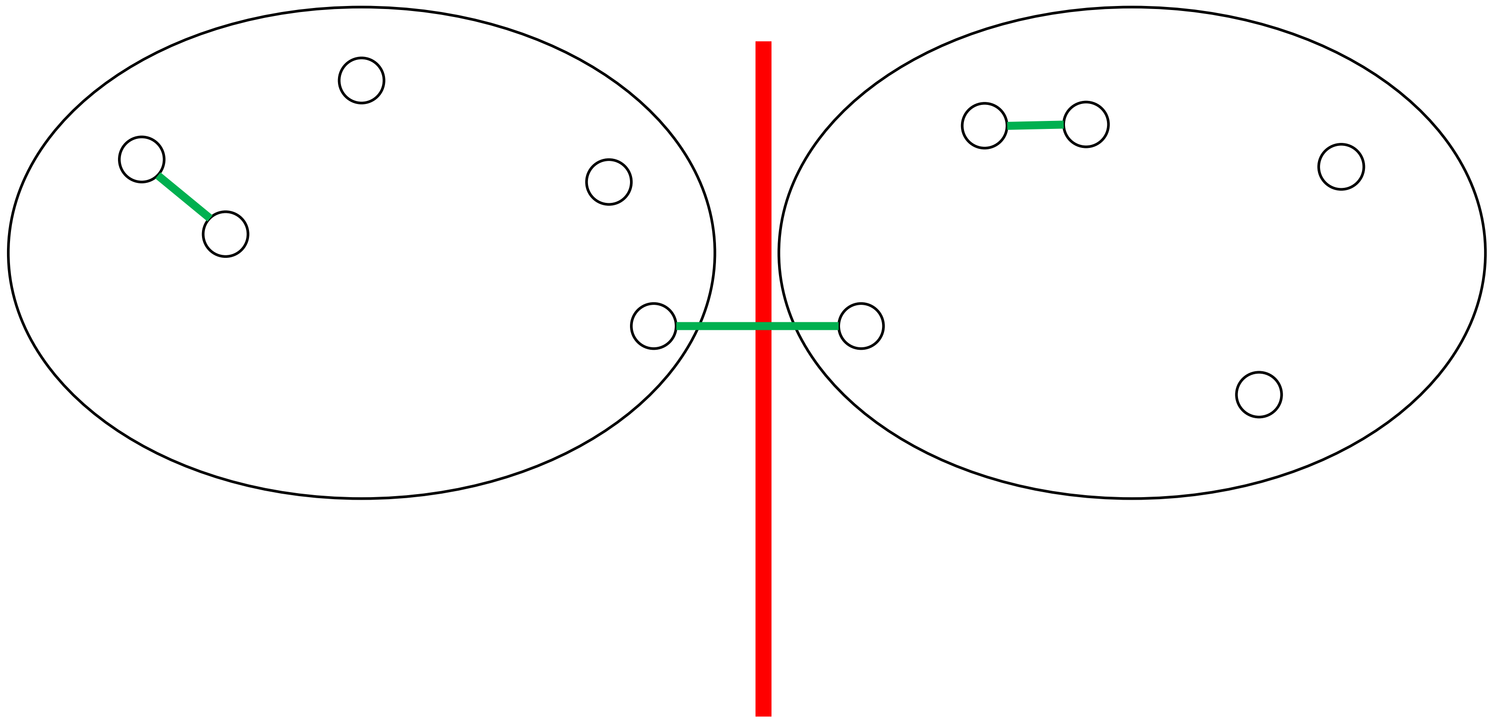
BOJ 1725 히스토그램

- 가운데 기둥을 지나는 모든 직사각형의 최대 넓이를 알면
- 나머지는 분할 정복 과정에서 구할 수 있음
- $T(n) = 2T(n/2) + O(n)$
- $T(n) = O(n \log n)$

```
ll N, A[101010];

ll Solve(int s, int e){
    if(s > e) return 0;
    int m = s + e >> 1;
    int l = m, r = m;
    ll now = A[m], h = A[m];
    while(l > s && r < e){
        if(min(h, A[l-1]) > min(h, A[r+1])) h = min(h, A[--l]);
        else h = min(h, A[++r]);
        now = max(now, h * (r-l+1));
    }
    while(l > s) h = min(h, A[--l]), now = max(now, h * (r-l+1));
    while(r < e) h = min(h, A[++r]), now = max(now, h * (r-l+1));
    return max({ now, Solve(s, m-1), Solve(m+1, e) });
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N; for(int i=1; i<=N; i++) cin >> A[i];
    cout << Solve(1, N);
}
```

BOJ 2261 가장 가까운 두 점

- 점들을 x좌표 오름차순으로 정렬하자.
- (분할) 적당한 d에 대해 x = d인 직선으로 분할해서 각각 해결
- (정복) x = d를 기준으로 반대편에 있는 점들의 쌍 고려



BOJ 2261 가장 가까운 두 점

- 정복을 대충 하면 $T(n) = 2T(n/2) + O(n^2) = O(n^2)$
- 정복을 잘 하면 $T(n) = 2T(n/2) + O(n\log n) = O(n\log^2 n)$
- 정복을 아주 잘 하면 $T(n) = 2T(n/2) + O(n) = O(n\log n)$

BOJ 2261 가장 가까운 두 점

- 분할에서 구한 양쪽의 정답 중 최솟값을 m이라고 하자.
- x = d를 지나는 선분 중 길이가 m 미만인 선분만 봐도 된다.

- x = d와 x좌표 차이가 m 미만인 점만 봐도 된다.
- 반대편에 있는 점은 y좌표 차이가 m 미만인 점만 봐도 된다.

- 각 점마다 최대 5개의 점만 보게 됨
  - Why?

BOJ 2261 가장 가까운 두 점

- x = d와 x좌표 차이가 m 미만인 점만 봐도 된다.
- 반대편에 있는 점은 y좌표 차이가 m 미만인 점만 봐도 된다.

- x = d와 x좌표 차이가 m 미만인 점을 모은 뒤 y좌표 정렬하고
- y 좌표 차이가 m 미만인 점을 쭉 보면 T(n) = O(n log^2 n)

- 현재 함수 Solve(s,e)가 종료되는 시점에
- 병합 정렬처럼 y좌표 정렬하면 T(n) = O(n log n)

BOJ 2261 가장 가까운 두 점

- n = 1이면 inf 반환(min 연산의 항등원)
- divLine은 점을 절반으로 나눌 수 있는 직선으로

```cpp
ll Dist(const Point &p1, const Point &p2){
    ll dx = p1.x - p2.x, dy = p1.y - p2.y;
    return dx * dx + dy * dy;
}

int N;
Point A[101010], T[101010];

ll DnC(int s, int e){
    if(s == e) return INF;
    int m = (s + e) / 2;
    ll divLine = A[m].x;
    ll d = min(DnC(s, m), DnC(m+1, e));

    int l = s, r = m+1, idx = s;
    while(l <= m && r <= e){
        if(A[l].y < A[r].y) T[idx++] = A[l++];
        else T[idx++] = A[r++];
    }
    while(l <= m) T[idx++] = A[l++];
    while(r <= e) T[idx++] = A[r++];
    for(int i=s; i<=e; i++) A[i] = T[i];

    vector<Point> now;
    for(int i=s; i<=e; i++){
        ll dx = A[i].x - divLine;
        if(dx * dx < d) now.push_back(A[i]);
    }

    ll ret = d;
    for(int i=1; i<now.size(); i++){
        for(int j=i-1; j>=0; j--){
            ll dy = now[i].y - now[j].y;
            if(dy * dy >= d) break;
            ret = min(ret, Dist(now[i], now[j]));
        }
    }
    return ret;

}
```

질문?

더 공부할 거리

- Centroid Decomposition
  - 트리 T의 경로 (u,v)에 대해 정의된 함수 f(u,v)
  - f(u,v)의 최대/최소/합 등등을 구하는 방법

  - 배열과는 다르게 절반으로 쪼개기 어려움
  - 모든 Subtree의 크기를 n/2 이하로 만드는 정점을 기준으로 분할

세그먼트 트리

구간의 합

- 배열 A에서 분할 정복으로 sum(A[l..r])를 구해보자.
  - 아래 코드에서 실제로 방문하는 (s, e) 순서쌍의 개수는 O(n)개
    - Why?
  - (s, e) 순서쌍에 대해 2번의 결과를 메모이제이션 or 전처리하면 어떨까?

```
int sum(int l, int r, int s=1, int e=N){
    if(r < s || e < l) return 0; // 1. 현재 구간 [s,e]에 쿼리 구간 [l,r]이 포함되지 않은 경우
    if(l <= s && e <= r){        // 2. 현재 구간 [s,e]가 쿼리 구간 [l,r]에 전부 포함되는 경우
        int ret = 0;
        for(int i=s; i<=e; i++) ret += A[i];
        return ret;
    }
    int m = (s + e) / 2;         // 3. 현재 구간 [s,e]와 쿼리 구간 [l,r]가 일부만 겹치는 경우
    return sum(l, r, s, m) + sum(l, r, m+1, e);
}
```

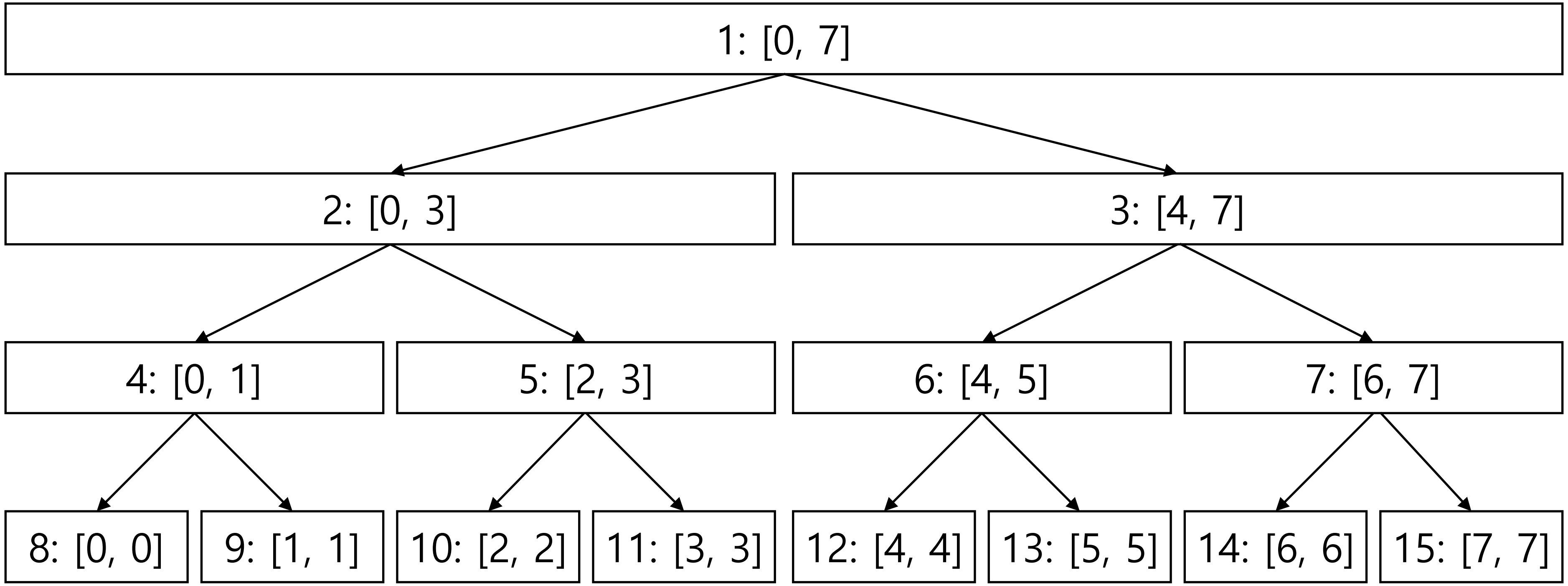
분할 정복 + 메모이제이션

- (s, e) 순서쌍에 대한 결과의 전처리는 O(n log n) 시간에 가능
  - (1, n) 구할 때 n
  - (1, n/2), (n/2+1, n) 구할 때 n
  - (1, n/4), (n/4+1, n/2), (n/2+1, 3n/4), (3n/4+1, n) 구할 때 n
  - …
  - (1, 1), (2, 2), (3, 3), … , (n, n) 구할 때 n

- 전처리가 끝나면 쿼리의 시간 복잡도는 O(log n)
  - 증명은 뒤에서…

세그먼트 트리

- 분할 정복을 메모이제이션하는 자료구조
- 포화 이진 트리 형태 : 배열로 관리할 수 있음
  - 부모 x/2, 왼쪽 자식 2x, 오른쪽 자식 2x+1



BOJ 2042 구간 합 구하기

- 구간 [s, e]를 관리하는 정점에 sum(A[s..e])를 저장
  - 사실 전처리는 O(n)에 가능
  - [i, i]를 관리하는 정점의 값을 A[i]로 정하고
  - 다른 정점들의 값을 아래부터 구하면 됨

- A[i]를 x로 바꾸는 쿼리
  - [i, i]를 관리하는 정점의 값을 x로 바꾼 뒤
  - 해당 정점의 조상들의 값을 갱신
- A[l]부터 A[r]까지의 합을 구하는 쿼리
  - 앞에서 봤던 분할 정복 코드 활용

BOJ 2042 구간 합 구하기

```
int N, M, K;
ll A[MAX_N], T[MAX_N * 4];

void Build(int node=1, int s=1, int e=N){
    if(s == e){ T[node] = A[s]; return; }
    int m = s + e >> 1;
    Build(node<<1, s, m);
    Build(node<<1|1, m+1, e);
    T[node] = T[node<<1] + T[node<<1|1];
}

void Update(int x, ll v, int node=1, int s=1, int e=N){
    if(s == e){ T[node] = v; return; }
    int m = s + e >> 1;
    if(x <= m) Update(x, v, node<<1, s, m);
    else Update(x, v, node<<1|1, m+1, e);
    T[node] = T[node<<1] + T[node<<1|1];
}

ll Query(int l, int r, int node=1, int s=1, int e=N){
    if(r < s || e < l) return 0;
    if(l <= s && e <= r) return T[node];
    int m = s + e >> 1;
    return Query(l, r, node<<1, s, m) + Query(l, r, node<<1|1, m+1, e);
}
```

세그먼트 트리 시간 복잡도

- Build : O(N)
  - 정점 O(N)개
- Update : O(log N)
  - 트리의 높이와 동일한 개수의 정점만 방문함
- Query : O(log N)
  - 진짜?
  - 트리의 각 level마다 최대 2개의 정점만 방문한다는 것을 보일 수 있음

세그먼트 트리 시간 복잡도

- case 1) 구간 안 겹침 : return 0;
  - 조건문을 잘 잡으면 이런 상황이 안 생기게 할 수 있음
  - 정점을 방문하지 않는다고 치자.
- case 2) 구간 전부 포함 : return T[node];
  - 정점을 방문해서 해당 정점의 값을 가져감
  - 추가로 다른 정점을 방문하지 않음
- case 3) 구간 일부 겹침 : 왼쪽, 오른쪽으로 재귀 호출
  - case a) [s, m]과 [l, r]만 겹치는 경우 자식 정점 하나 방문
  - case b) [m+1, e]와 [l, r]만 겹치는 경우 자식 정점 하나 방문
  - case c) 둘 다 겹치는 경우 자식 정점 2개 방문

세그먼트 트리 시간 복잡도

- case 2 : 현재 level에서 방문하는 정점 개수 1 증가
- case 3-a : 현재/다음 level에서 방문하는 정점 개수 1 증가
- case 3-b : 현재/다음 level에서 방문하는 정점 개수 1 증가
- case 3-c
  - 현재 level에서 방문하는 정점 개수 1 증가
  - 다음 level에서 방문하는 정점 개수 2 증가
  - 최대 1번 발생
- 각 level마다 최대 2개의 정점만 방문함 : O(log N)

질문?

비재귀 세그먼트 트리

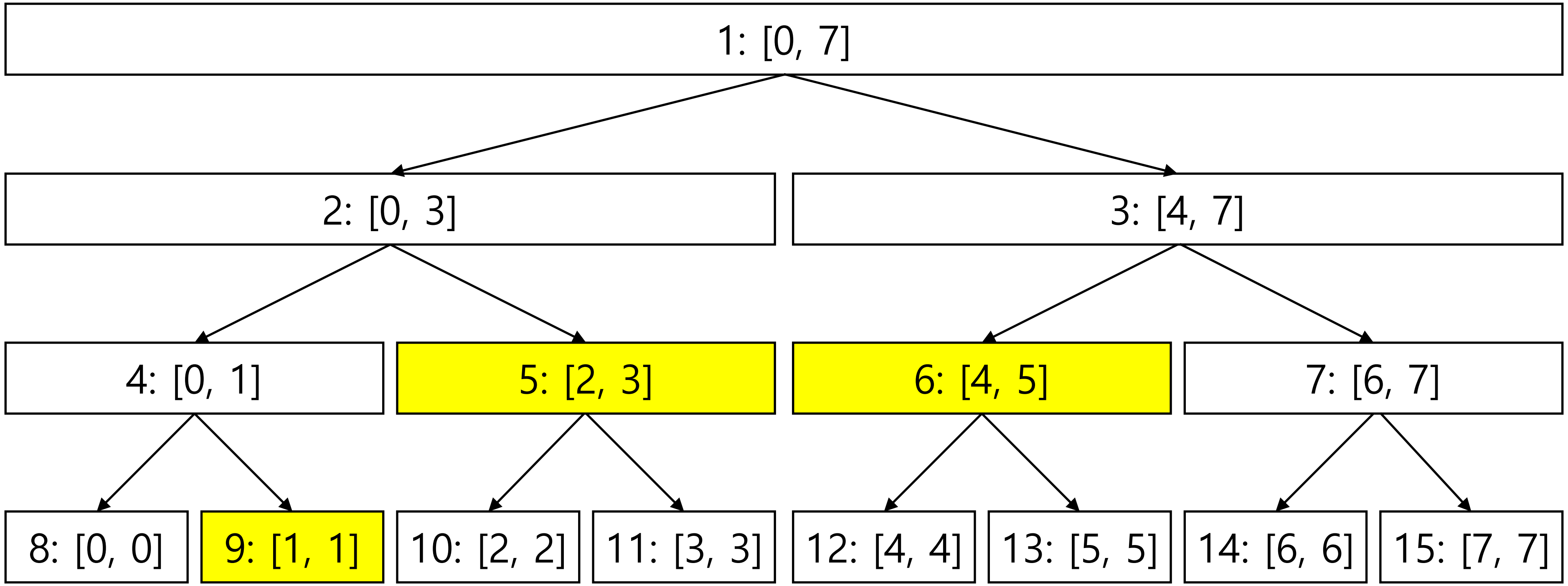
- 재귀 대신 반복문을 사용하면 속도가 더 빨라짐
  - 관리하는 구간을 [0, 2^k - 1]로 고정하자.
  - 리프 정점의 번호는 2^k ~ 2^(k+1)
    - [i, i]를 관리하는 리프 정점의 번호는 2^k + I
  - 내부 정점의 번호는 1 ~ 2^k-1

  - Build랑 Update는 구현 쉬움
  - Query는?

```
constexpr int SZ = 1 << 20;

int N, M, K;
ll A[SZ], T[SZ << 1];

void Build(){
    for(int i=1; i<=N; i++) T[i|SZ] = A[i];
    for(int i=SZ-1; i; i--) T[i] = T[i<<1] + T[i<<1|1];
}

void Update(int x, ll v){
    x |= SZ; T[x] = v;
    while(x >>= 1) T[x] = T[x<<1] + T[x<<1|1];
}
```

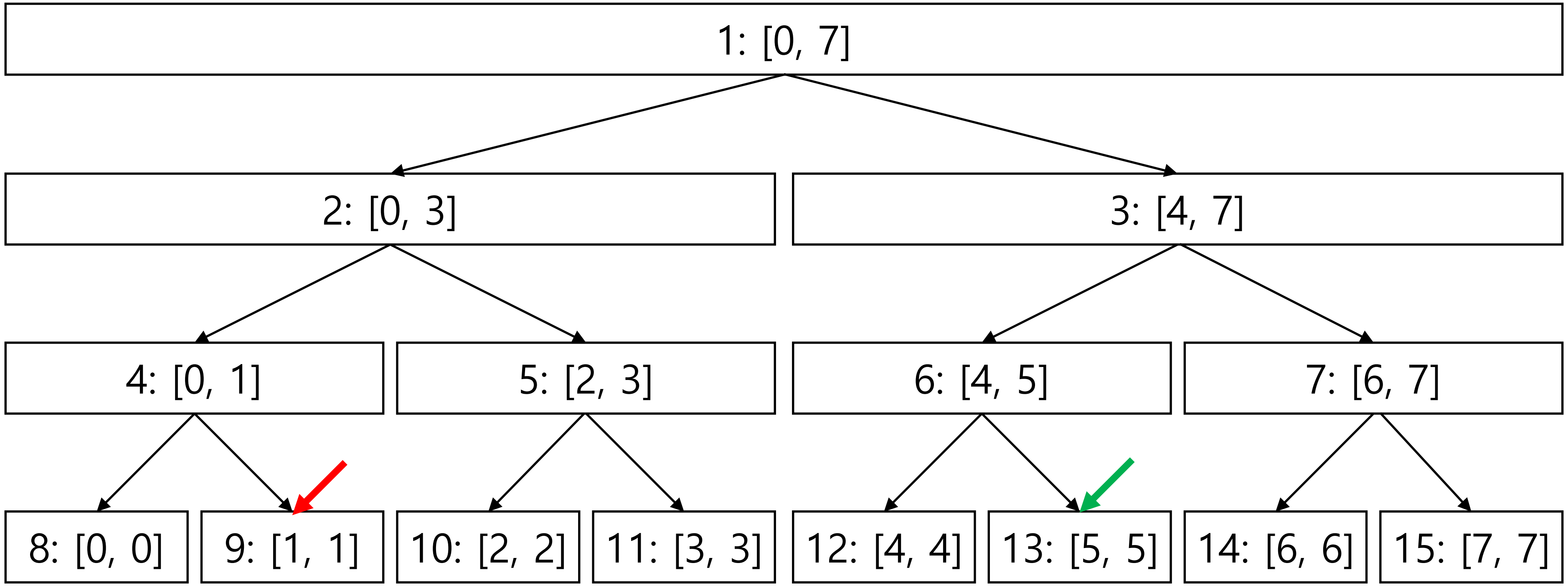
비재귀 세그먼트 트리

- [1, 5]의 구간 합을 구해보자.



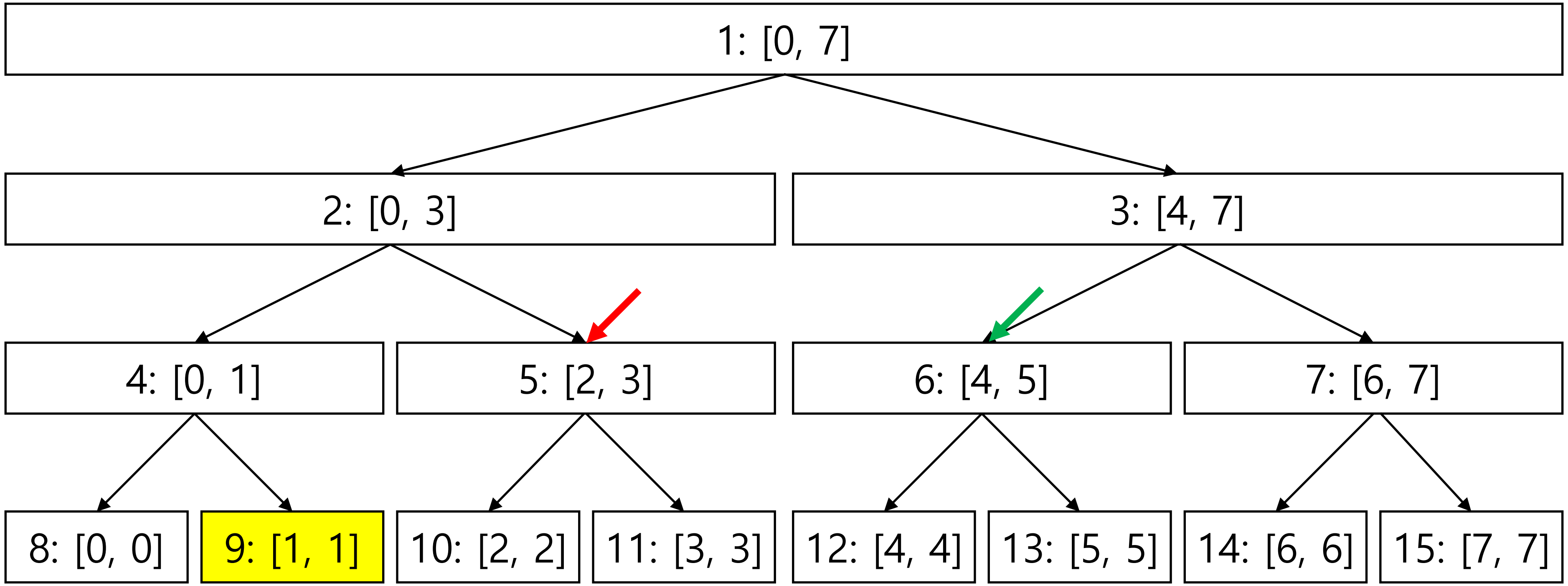
비재귀 세그먼트 트리

- [1, 5]의 구간 합을 구해보자.
  - 1번째 리프 정점과 5번째 리프 정점에서 시작한다.
  - 각각 l, r이라고 하자.



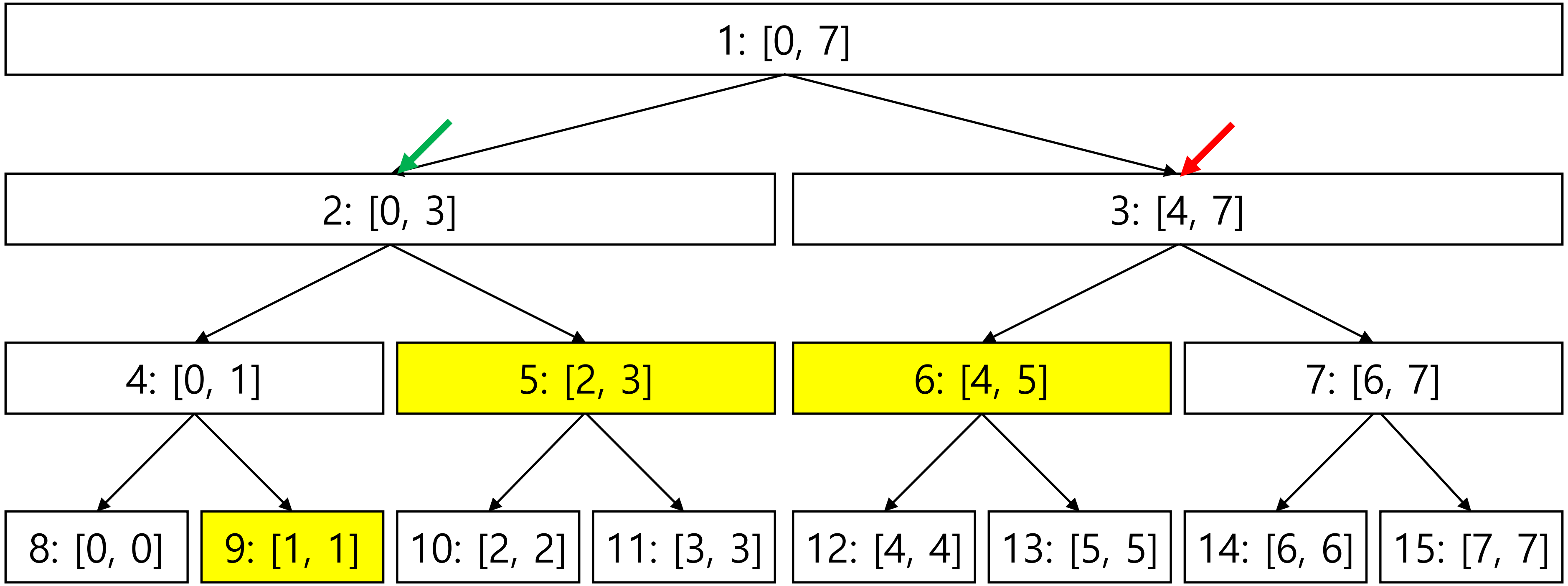
비재귀 세그먼트 트리

- [1, 5]의 구간 합을 구해보자.
  - l은 오른쪽 자식이다. l의 부모는 [1,5]에 포함되지 않으므로 l의 값을 결과에 반영
  - r은 오른쪽 자식이다. r의 부모는 [1,5]에 포함되므로 r의 부모만 고려해도 됨
  - l은 1 증가시킨 뒤 한 칸 위로, r은 그냥 한 칸 위로



비재귀 세그먼트 트리

- [1, 5]의 구간 합을 구해보자.
  - l은 오른쪽 자식이다. l의 부모는 [1,5]에 포함되지 않으므로 l의 값을 결과에 반영
  - r은 왼쪽 자식이다. r의 부모는 [1,5]에 포함되지 않으므로 r의 값을 결과에 반영
  - l은 1 증가시킨 뒤 한 칸 위로, r은 1 감소시킨 뒤 한 칸 위로 (l>r이므로 종료)



비재귀 세그먼트 트리

- [L, R]의 합을 구한다고 하자.
  - L번째 리프 정점과 R번째 리프 정점에서 시작한다. l, r이라고 하자.
  - l ≤ r 이면 아래 과정 반복
    - l의 정점 번호가 홀수인지 짝수인지 확인
      - l이 짝수이면 왼쪽 자식
        - l의 부모 정점이 [L,R]에 온전히 포함되므로 l의 부모만 고려해도 됨 (l /= 2)
      - l이 홀수이면 오른쪽 자식
        - l의 부모 정점은 [L,R]에 포함되지 않으므로 l의 값을 결과에 반영하고 오른쪽으로 이동 (ret += T[l++])
        - 해당 정점은 왼쪽 정점이므로 부모가 [L,R]에 온전히 포함됨. 부모만 고려해도 됨 (l /= 2)
    - r의 정점 번호가 홀수인지 짝수인지 확인
      - r이 홀수이면 오른쪽 자식 (r /= 2)
      - r이 짝수이면 왼쪽 자식 (ret += T[r--], r /= 2)

비재귀 세그먼트 트리

```
constexpr int SZ = 1 << 20;

int N, M, K;
ll A[SZ], T[SZ << 1];

void Build(){
    for(int i=1; i<=N; i++) T[i|SZ] = A[i];
    for(int i=SZ-1; i; i--) T[i] = T[i<<1] + T[i<<1|1];
}

void Update(int x, ll v){
    x |= SZ; T[x] = v;
    while(x >>= 1) T[x] = T[x<<1] + T[x<<1|1];
}

ll Query(int l, int r){
    l |= SZ; r |= SZ;
    ll ret = 0;
    while(l <= r){
        if(l & 1) ret += T[l++];
        if(~r & 1) ret += T[r--];
        l >>= 1; r >>= 1;
    }
    return ret;
}
```

K번째 수 구하기

- 배열 A에서 K번째로 작은 수를 구하는 문제
  - 임의의 K에 대해 여러 번 구해야 한다면?
  - 원소가 추가/삭제된다면?

- 구간 합을 관리하는 세그먼트 트리로 O(log N)에 가능
  - 현재 정점 x에서 2가지 경우로 나뉨
    - case 1) K ≤ T[x*2] : K번째 수가 왼쪽에 있음
    - case 2) K > T[x*2] : K번째 수가 오른쪽에 있음

K번째 수 구하기

```
int N, T[SZ << 1];

void Update(int x, int v){
    x |= SZ; T[x] += v;
    while(x >>= 1) T[x] += v;
}
int Kth(int k){
    int x = 1;
    while(x < SZ){
        if(k <= T[x << 1]) x = x << 1;
        else k -= T[x << 1], x = x << 1 | 1;
    }
    return x ^ SZ;
}
```

질문?